Abstract

In the area of network security, there are numerous tools available for monitoring and for vulnerability scanning, but each has a unique way of representing its results. These log files, sometimes megabytes in size, can take hours or days for a system administrator to wade through. Although it is possible to develop tools to find relationships between events in a single log and even between events in multiple logs from the same tool, there currently is no way of searching for these relationships between different logs from different security tools. This project takes the first step towards this goal by providing (a) a single relational database in which each tool's log files will be stored and (b) a mechanism for routinely updating the database with the latest data from these security tools. In the future, this homogeneous format for storing security-related tools' outputs may be used for trend analysis and other data mining techniques in order to discover otherwise obfuscated events.

Key Sources

DARPA Common Intrusion Detection Framework. http://seclab.cs.ucdavis.edu/cidf/

Feiertag, Rich, Cliff Kahn, Phil Porras, Dan Schnackenberg, Stuart Staniford-Chen, and Brian Tung, ed. "Common Intrusion Specification Language," DRAFT Specification, 8 June 1998. http://gost.isi.edu/projects/crisis/cisl_0.7.txt

Concordia Mobile Agent System. http://www.meitca.com/HSL/Projects/Concordia/Welcome.html

About the Author

Robert A. Mixer is an officer in the U.S. Air Force. He was a distinguished graduate of the U.S. Air Force Academy in 1997 and received his Master of Computer Science degree at Texas A&M University in 1998 in the area of network security.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 26.Jul.99	3. REPORT TYPE AND DATES COVERED THESIS

4. TITLE AND SUBTITLE COMMON DATABASE FORMAT FOR NETWORK SECURITY DATA	5. FUNDING NUMBERS
6. AUTHOR(S) 2D LT MIXER ROBERT A	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) TEXAS A&M UNIVERSITY	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) THE DEPARTMENT OF THE AIR FORCE AFIT/CIA, BLDG 125 2950 P STREET WPAFB OH 45433	10. SPONSORING/MONITORING AGENCY REPORT NUMBER FY99-186
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION AVAILABILITY STATEMENT Unlimited distribution In Accordance With AFI 35-205/AFIT Sup 1	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

DISTRIBUTION STATEMENT A
Approved for Public Release
Distribution Unlimited

14. SUBJECT TERMS			15. NUMBER OF PAGES 25
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

COMMON DATABASE FORMAT FOR NETWORK SECURITY DATA

by

Robert A. Mixer (0465)

A thesis submitted to the Faculty of Texas A&M University in partial fulfillment of the requirements for the degree of Master of Computer Science

Summer 1998

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

In the area of network security, there are numerous tools available for monitoring and for vulnerability scanning, but each has a unique way of representing its results. These log files, sometimes megabytes in size, can take hours or days for a system administrator to wade through. Although it is possible to develop tools to find relationships between events in a single log and even between events in multiple logs from the same tool, there currently is no way of searching for these relationships between different logs from different security tools. This project takes the first step towards this goal by providing (a) a single relational database in which each tool's log files will be stored and (b) a mechanism for routinely updating the database with the latest data from these security tools. In the future, this homogeneous format for storing security-related tools' outputs may be used for trend analysis and other data mining techniques in order to discover otherwise obfuscated events.

1. INTRODUCTION

Network security has received growing interest as an increasing number of businesses and institutions establish local area networks and connect with the rest of the Internet. Corporations want to provide important information to customers and potential customers about their products and other corporate information while protecting their most vital information from tampering. Because networking was originally designed as a completely open environment, however, securing private information has been added as an afterthought.

The security tools that have emerged as a result of the push to prevent unwanted access to private information have not adhered to any standard way of operating because none exists. The problem, then, involves incorporating several specialized security tools into one *corporate security package* and getting them to work synergistically. Because most were not designed to work with other security tools and their reports do not follow any standard format, each typically executes in isolation and requires an administrator's watchful eye. As any system administrator who has waded through a security tool's report stacked six feet high can attest, simply detecting intrusions can become more than a full-time job.

In order to combat the problem of inconsistent output formats from these security tools, this project involved developing a single database format for them. To make such a product useful, I've also developed an entire mobile agent system to (1) retrieve data from the security tools, (2) to update the database with the new information, and (3) to visually design the parser so that the administrator has complete control over the parsing process.

The benefits of having a common format for this security data are enormous because of the possibilities it presents. First, trend analysis (detecting trends across time regardless of the security tool that detected the events) becomes possible. Collaboration between different security tools becomes possible as well. This is particularly important because what each tool detects individually may not signal an intrusion to either one, but the combination of the two results may indeed indicate one. Finally, one can design other agents to update this database with the information it is missing. This can help complete the network state information for a given point in time in order to help draw conclusions that are more accurate.

2. APPROACHES

This problem of designing a common format for network security tool's output is just now receiving some attention. There is one other group currently attempting to specify a common format for the same purposes, although their approach does not involve a relational database. This section will briefly review their approach and an approach that I took earlier in this project's preparation.

2.1. DARPA Common Intrusion Detection Specification Language

Teresa Lunt initiated this effort at the Defense Advanced Research Projects Agency (DARPA) in order to help security tools be able to work together. To date, their DARPA Common Intrusion Detection Framework now includes both a specification language and a communications protocol. This section will give an overview of their specification language and assess its advantages and disadvantages.

2.1.1. Description

The specification language to date attempts to be expressive, unique in expression, precise, layered, self-defining, efficient, extensible, simple, portable, and easy to implement. To meet these goals, those working on this project have proposed the use of S-expressions such as (hostname ‘neuron.cs.tamu.edu’) that can be nested, can represent causal relationships, can take a list of values, or can take a single value. The example below is an example that they provide in order to help understand how these S-expressions are built to represent a sequence of events:

```
(InSequence
  (Login
    (Context  (Time '14:57:36 24 Feb 1998'))
    (Initiator (HostName 'big.evil.com'))
    (Account (UserName 'joe')
             (RealName 'Joe Cool')
             (HostName 'ten.ada.net')
             (ReferAs 'FauxJoe')))
  (Delete
    (Context (HostName 'ten.ada.net')
             (Time '14:58:12 24 Feb 1998'))
    (Initiator (ReferTo 'FauxJoe'))
    (Source (FileName (ExtendedBy UnixFullFileName)
                      '/etc/passwd')))
  (Login
    (CriticalContext
       (ReturnCode (ExtendedBy CIDFReturnCode) Failed)
       (Comment '/etc/passwd missing'))
    (Context (Time '15:02:48 24 Feb 1998'))
    (Initiator (HostName 'small.world.com'))
    (Account (UserName 'mworth')
             (RealName 'Mary Worth')
             (HostName 'ten.ada.net')))
)
```

The English translation they provide for this S-expression follows: “Three actions took place in sequence, at the times indicated. First, someone logged into the account named 'joe' (real name 'Joe Cool') at 'ten.ada.net', from a host named 'big.evil.com'. Then, about a half-minute later, this same person deleted the file '/etc/passwd' from 'ten.ada.net'. Finally, about four-and-a-half minutes later, a user attempted but failed to log in to the account 'mworth' at 'ten.ada.net'. The attempted login was initiated by a user at 'small.world.com'."

From this example, one can see that this language is very expressive and appears quite simple. Their specification further defines the syntax for building these expressions so that they follow a similar format and so that another tool can interpret what these S-expressions mean. It also defines an encoding scheme to decrease the memory footprint of an S-expression because storage in ASCII is quite wasteful.

2.1.2. Advantages and Disadvantages

Although the previous discussion did not discuss all of the details of this particular language specification, one can get a good feel for the language by reviewing the example that they provide in their documentation. The language is *very* expressive and extensible. Another major advantage is that the LISP-like S-expressions make it easy to implement converters to and from this language. For collaboration or binary storage, this particular format is already looking very promising.

The main disadvantage of this language is its complexity if implemented in a relational database. Due to the inherent nested structure and expressiveness, a database that provided the ability to store and retrieve these S-expressions would be extremely complex. If we were to implement a relational database to represent *just* the example S-expression above, the database would require 11 tables for the entities and at least 6 supporting tables to allow for the 1:N relationships between some of the entities.

2.2. CLIPS Security Format

During the early stages of this project, I pursued a similar approach to the DARPA approach (unknowingly) in order to provide some of the same expressiveness to the language [4]. My goals at the time were similar to theirs in the respect that the common data format should be as nearly expressive as the original security tool's output while still removing the ambiguities of natural language processing. I also was not attempting to create a language that would map readily to relational database representation, but examining this additional approach should help provide a better feel for the trade-offs involved in creating a common format for network security data. This section will give an overview of this initial design and discuss its advantages and disadvantages.

2.2.1. Description

Borrowing from the C Language Integrated Production System (CLIPS) [5] use of templates for facts, I specified the following templates for use in the system.

```
<data-fact> ::= (data (source <source>) (time <time>)
      (date <date>) (host <host>) (area <area>) (keywords
      <keywords>) (certainty <certainty>))

<summary-fact> ::= (summary (area <area>) (certainty
      <certainty>) (items <items>))
```

In addition, the syntax for the field values:.

```
<time> ::= HHMMSS where 00 <HH < 24

<date> ::= YYYYMMDD

<object> ::= group_file | group | password_file | object
      | path | user | password | umask| home_dir | root-
      path | world

<attribute> ::= missing | blank | guesses | duplicate |
      nonnumeric | readable | writable | negative |
      exported
```

```
<value> ::= <number> | <string>

<keywords> ::= [<object> <value> | <attribute> <object>]*

<items> ::=  <value>* | <object> <value>*

<certainty> ::= <float> range 0.0 to 1.0
```

2.2.2. Advantages and Disadvantages

This syntax proved to be able to express every possible output from the COPS program [6] without losing anything important. It also proved to be very useful within a CLIPS inference engine because rules did not need to specifically match every field within the template. In fact, the only fields that needed to be specified for a particular rule were those that were needed for the rule to fire. Since the application in which these facts were used relied heavily on the quickness of rule matching and firing, the syntax accomplished its goals.

However, there are some drawbacks to this specification language. First, while it may provide enough expressiveness for the COPS security tool, it does not necessarily provide enough expressiveness for every security tool. It certainly cannot express as completely the example that the DARPA specification language could. It also is difficult to deduce exactly what the original 'English' entry stated because of the structure of the <keywords> object. *This language is limited to specifying who was involved and what special attributes applied to this entry*, but it does not contain any ability to represent sequences of actions. Finally, implementing this specification language in a relational database would be extremely complex because of the large number of possible values for <object> and the 1:N mapping from <keywords> to a <fact>.

4. DESIGN

In light of these two different languages for expressing security data, I developed a common database format to accomplish the same goal. In order to prevent the database's complexity from growing too high, I chose to limit the expressiveness of this common format. While still being able to represent all that the CLIPS Security Format could represent, this common database format currently cannot represent all that the DARPA Common Intrusion Detection Specification Language can. The design of the database and the supporting mobile agent system allows for easy extensions to this system to be built in order to provide greater expressiveness. This section will discuss the design of both the database format and the supporting mobile agent system.

4.1. Database Format

The main table in the database is the Problem table that assigns a unique ID to a new Problem, an Area to the problem, and stores the original tool's output for this particular problem. The storage of the original ASCII output allows for future expansion because one can retrieve the entry exactly as it appeared in the tool's output and retrieve additional data if necessary. Although ASCII storage of long entries can require a lot of disk space, its provision for future expansion far outweighs the cost. One additional feature of this ASCII storage is that it is very quick to check whether this problem has been previously reported. If so, the database only needs to store a new entry in the ProblemTime table.

The ProblemTime table contains the rest of the 'context' information by recording the host it was reported on, the time of detection, and the source (security tool) which generated this entry. As mentioned, for problems that have been reported before, we only need to add another entry in this table with the same Problem ID and just add the new time, host, and source data. This provides considerable storage savings over storing the same ASCII text for each occurrence of the same event.

The rest of the tables are merely supporting tables containing information about objects in the network that are not time-dependent. There is a File table, Directory table, Group table, User table, Host table, and Area table (to categorize each problem). Finally, a set of intermediate tables maps objects in the network and any time-dependent information with a specific Problem. There exists one of these for Problem-File relationships and Problem-Directory relationships, Problem-Group and Problem-User relationships, and one for Problem-Other relationships. This Problem-Other table can be used to add additional expressiveness to any problem entry by associating a name-value pair with the problem even when the database does not contain a table for that specific object.

The diagram on the next page shows the three levels of tables in the database. Those on the left are the Object tables, those in the middle are the Mapping tables, and those on the right are the top-level Problem tables. The relationships between attributes in the tables are shown in the diagram as well.

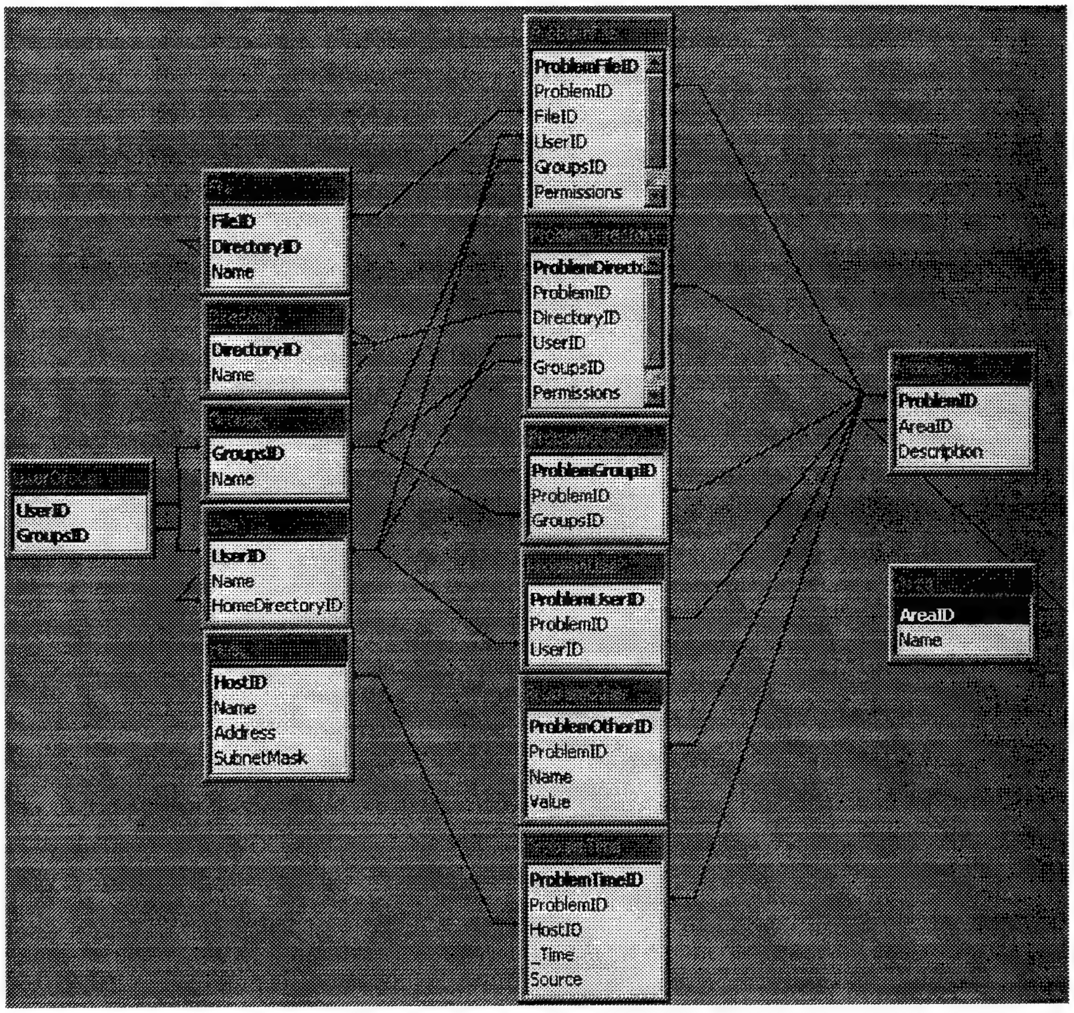


Figure 1: Database tables and their relationships

4.2. Supporting Mobile Agent System

The second and more difficult portion of the project was to develop a system to update this database with new security data as it became available. In order for the database to be easily extensible, this update system must itself be extensible. This section details the use of a mobile agent system for this task and the design of the components within this system.

4.2.1 Concordia Mobile Agent System

I chose a mobile agent architecture written entirely in Java for two reasons. First, Java would provide platform independence so that the addition of security tools that reside on other platforms would be less difficult. Second, a mobile agent system provides many advantages over the single application approach as this section will describe. Of all the ones available, the Concordia Mobile Agent System [3] developed by Mitsubishi Electric Information Technology Center America is the best in the areas of (1) ease of development and use, (2) footprint on the local host and in embedded code, and (3) implementation of security features throughout its product.

The benefits of using a mobile agent system for such an application can only be appreciated by comparing it with a single application designed to perform the same task. Mobile agent systems provide significant advantages in the area of bandwidth utilization and CPU utilization.

First, the mobile agent system allows us to place the database-intensive code right on the same host as the database. In fact, we can place all of the CPU intensive services such as database-update and security-tool-output-parsing services right on the host on which they will be performing the function. Once running, these services do not need to do any traversal across the network. A single application has the same advantage of preventing its code from having to traverse the network, but its disadvantage is that all traffic to specific hosts (including database access) must travel across the network. Since security tool's output files can be megabytes in size, this would require a lot of traffic (from host to application and then from application to database). This distribution of CPU use in mobile agents greatly eases the burden from any single host, and the reduction in network traffic eases the burden on the network.

Second, mobile agents are very logically mapped to the task. First, we send an agent to a site to collect the newly parsed security tool data. The agent then travels to the database service host to update the database. Finally, it returns to the sender to report its success or failure. In contrast, a single application must download the security tool's output, parse it, and perform database queries and updates from across the network. In a corporate environment, we would send a courier to collect processed data and deliver it to the client who requested it – why not have our software model the same process?

The only drawback to any security system is the security of the information that is passed and the mutual authentication of the entities. The commercial version of the Concordia Mobile Agent System contains a feature-rich security framework complete with access control lists, signed archives of files, and encryption of agents during transmission and persistent storage. For this project, however, the educational version (which lacks these security features) provided the remainder of the features needed to transport agents throughout the network and demonstrate the feasibility of the concept.

4.2.3. High-level design

The highest level design can be broken into two parts: the agents and the services, both of which were almost trivial given the low-level design discussed in the next section. In addition to these two components, I've added a graphical user interface to construct and deliver new sets of parser rules to specific services.

4.2.3.1. Services

The services are simply a collection of SecurityToolInterface services, a Directory service, and a SecurityDB (Security Database) service. The SecurityToolInterface services are the ones that interact directly with a specific security tool at a specific host. When an agent arrives and tells it to retrieve data, this service is responsible for obtaining any available output from the security tool, parsing it, and providing the parsed results to the agent for transport to the SecurityDB service. The SecurityToolInterface services also provide a method for updating the parsing rules that it uses in order to allow for future updates to the security tool.

The SecurityDB service resides as close as possible (network proximity) to the host on which the physical database resides. When an agent arrives with new parsed security results, the SecurityDB service uses these results to query and update the database. The parsed results are always partially incomplete because primary key values for the different tables are not known to the distributed SecurityToolInterface services. Thus, the SecurityDB service updates these parsed results as it obtains the needed information from the data already in the database. The next section will discuss this process in more detail because the lower-level abstraction hides all of the messiness of the dependencies between parsed result entries.

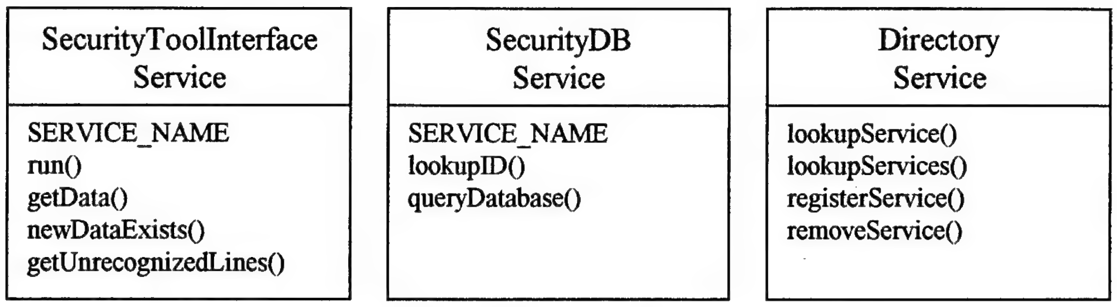


Figure 2: Services in the mobile agent system

Finally, the Directory service provides a service-to-host mapping for agents to query. Rather than specifying a specific itinerary for an agent, an agent may query the Directory service for one or all of the hosts at which a particular SecurityTool-Interface service resides and then travel to these hosts as needed. This service is a very simple service that uses Java's Remote Method Invocation (RMI) capabilities in order to avoid having agents travel to the host on which this service resides. The RMI protocol is similar in functionality to the Remote Procedure Call (RPC) protocol [7].

4.2.3.2. Agents

The agent types in this system are very few: a CronAgent, a CheckParserRulesAgent, and a UpdateParserRulesAgent. The CronAgent is the courier mentioned thus far, written as a stand-alone utility for routinely checking all sites registered as SecurityToolInterface sites. The agent forks off a separate CronAgent for each check. This agent then travels to the SecurityToolInterface site and requests if new data exists. If so, it instructs the SecurityToolInterface to parse the data, and then the agent travels to the database to update it. This tool provides the ability to log all actions to a log file in order to track what it is doing. Their bandwidth requirement is almost completely comprised of the parsed data that they retrieve and deliver because the remainder of their code is just an itinerary.

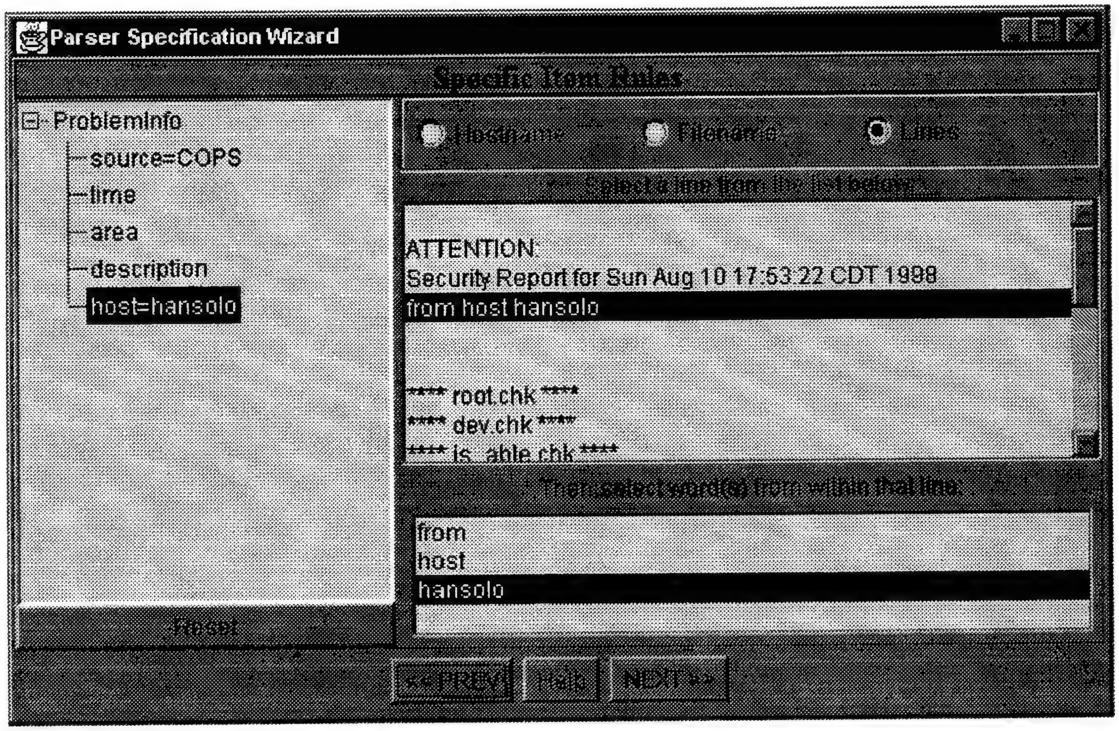
The second agent is the UpdateParserRulesAgent responsible for carrying a new set of ParserRules to individual SecurityToolInterface services. The bandwidth requirement for this agent is also almost completely comprised of its payload. The Java Parser Specification Wizard that I developed is responsible for creating agents of this type and sending them out when a user has completed the parser specification process (more on this in the next section).

The final agent is the CheckParserRulesAgent that merely travels to a designated SecurityToolInterface service site and retrieves its parser rules. This agent is best used to verify that ParserRules actually arrived when the UpdateParserRulesAgent delivered them.

4.2.3.3. Java Parser Specification Wizard

The final portion of the high-level design is the Parser Specification Wizard that an administrator can use to completely customize exactly how a particular parsing process is done. This process is simplified into a series of steps in a familiar 'wizard' format. A sample screen shot of this wizard is shown on the next page.

The wizard is unique in the respect that as parser rules are added to the system, the user can watch to see how the parser applies the rules to subsequent lines in the file. Only when a line is reached that is not recognized by current parsing rules does the wizard prompt the user to specify how to interpret the line.


Parser Specification Wizard
Specific Item Rules
Hostname
Filename
Lines
Select a line from the list below:
Then select word(s) from within that line:
Reset
<<PREV
Help
NEXT>>

```
ProblemInfo
  source=COPS
  time
  area
  description
  host=hansolo
```

```
ATTENTION:
Security Report for Sun Aug 10 17:53:22 CDT 1998
from host hansolo


**** root.chk ****
**** dev.chk ****
**** is_able.chk ****
```

```
from
host
hansolo
```

Figure 3: Screen shot of Java Parser Specification Wizard

At the end of the specification process, the wizard prompts the user for the type of SecurityToolInterface service that this particular rule set applies to and sends a ParserRulesUpdate agent to each host where that service resides. Thus, an administrator can dynamically change how a parser interprets output from a security tool and an administrator can design a parser for a security tool that the current implementation does not support. The administrator merely needs to copy an existing SecurityToolInterface service implementation, modify its name, and then use this wizard to specify how it should interpret the output from its corresponding security tool.

The only drawback to using such a specification process is that more than likely the parser will not be complete because the output file used to build it did not contain every possible output the security tool is capable of generating. Since most security tools do not specify all of their possible outputs, this wizard provides a good approximation to a complete parser. To evaluate how complete the parser is, an agent can query the SecurityToolInterface service after parsing to retrieve all of the lines that were not recognized by the parser.

4.2.4. Low-level design

Although the high-level design is exciting, the low-level design is what makes the high-level design possible. The design itself centers upon three different layers of abstraction: the DBClass layer, the DBInfo layer, and the DBInfoClass layer. These layers are shown below.

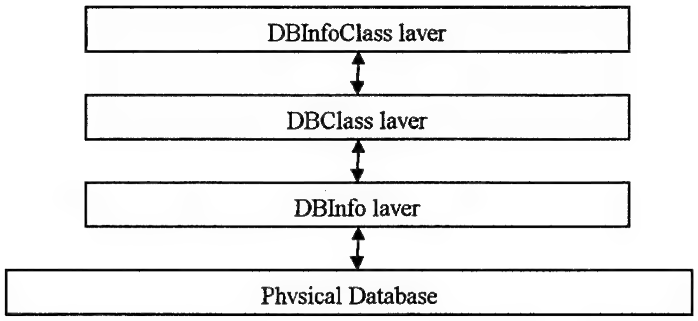


Figure 4: Layers in the low-level design

4.2.4.1. DBClass Layer

The DBClass layer is the layer most closely related to the actual database tables. In fact, each DBClass subclass must be identical to the table that it represents in name and in the attributes it contains. For example, the Problem table in the database contains the attributes ProblemID(long), AreaID(long), and Description(String); and so its corresponding DBClass Problem contains the attributes ProblemID(long), AreaID(long), and Description(String). This is necessary because we can use Java's Reflection packages [8] to dynamically determine for any given DBClass just what we need to update in the database. Given a Problem DBClass, the DatabaseService can use the class's name to know the table to update and cycle through its attributes to know what to set each attribute to in the table. For security's sake, access to these attributes in a DBClass is only directly visible to classes meeting a customizable security criteria (signature or equivalent).

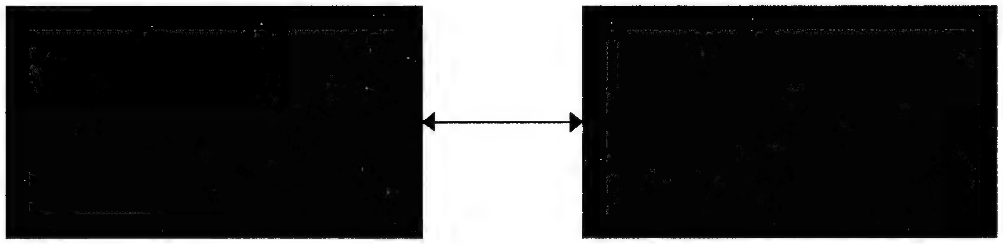


Figure 5: Correspondence between database table and DBClass class

4.2.4.2. DBInfo Layer

A parser shouldn't need to mess with setting IDs and other such values, nor should it need to concern itself with dependencies between attributes in different tables. This is the purpose for the next level of abstraction: the Info layer. The DBInfo layer is a collection of classes that group similar information together. For example, ProblemInfo groups the source, time, host, area, and description all together into one class. Each of these attributes is a String so that a parser can readily set the value of an Info field with the ASCII it retrieves from a file.

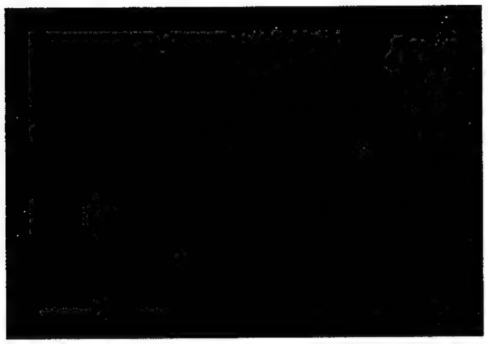


Figure 6: DBInfo class ProblemInfo

The internal workings of the Info layer classes is what does the work of transforming the String attribute values into the DBClass layer classes that the SecurityDB service needs. Each has a method to convert its attributes into DBClass layer classes and to specify the dependencies between them, but all of this is hard-coded into the DBInfo layer class. The actual transformation takes place at the SecurityDB service when it invokes that transformation method. Then the SecurityDB service simply cycles through the given DBClass layer classes, updating the database with independent classes at each cycle until no more are left. The dependent classes receive their values as the independent classes are added to the database in order to avoid an infinite loop.

4.2.4.3. DBInfoClass Layer

At the highest level of abstraction is the DBInfoClass layer. Although the name is somewhat confusing, this set of classes serves its purpose in the parser and in parser specification. A parser rule needs to know what keywords to look for in what locations in a line and what specific items map to what Info class attributes. It also needs to know what filters to apply to an item that it retrieves from a line because the item in its raw form may not be meaningful in the database. Consider the following COPS output line and the significant values an administrator would note for the line:

```
Warning!  User foo's home directory /user/foo is mode 0777!
```

DBClassInfo representation:
User.Name found at word 3 in the line
Directory.Name found at word 6 in the line
Directory.Permissions found at word 9 in the line

Figure 6: Mapping from line to DBClassInfo

A parser needs to know that item 3 ("blah's") is the item to map to the user name and that it needs to apply a substring filter in order to arrive at "blah." The same holds true for the directory name and the mode. The rest of the words serve as keywords to identify this particular rule. The DBInfoClass layer provides this mapping of keyword-index pairs, filters, and Info-index pairs for ParserRules.

5. PERFORMANCE

Because Java has a reputation for being slower than its counterparts C and C++, I attempted to optimize the parsing process and database update process in every way.

In order to speed performance during parsing, I've implemented a simple rule cache to store the most recently applied rules. I noticed that security tools generally check things in similar groups. For example, COPS checks for permissions problems, and then for user problems, and then for password problems, etc. Because of this locality, I found that just putting a simple cache with four entries reduced the search time for applying rules by 70% or more! The parser just checks in the cache when a new line is encountered before doing a linear search through its rules to see which one (if any) applies.

I also added the ideas of IgnoreRules and delimiters to help weed out lines that are not true security tool entries. Many tools print headers between different sections of their output that should just be ignored. Likewise, most preface their actual security tool entries with a delimiter like "Warning!" or "Error!". Applying these two ideas when lines are read from the file prevents unnecessary searches through the parser rules.

The most tedious process is the Parser Specification Wizard process because an administrator must single step through every line of a security tool's output in order to specify keywords and item mappings for each entry. Ironically, the most accurate parsing results are obtained when an administrator uses a file from the security tool that contains a variety of entries because the parser is then the most complete.

6. CONCLUSION

The Common Database Format for Network Security Data and its supporting mobile agent system is a pioneering effort towards developing a complete common intrusion detection framework. Although it is not as expressive as other intrusion detection specification languages, it does provide reasonable expressiveness and, more importantly, it has been implemented and tested. This project lays the foundation for future trend analysis and collaborative intrusion detection, and can easily be extended to provide more expressiveness in the future.

REFERENCES

[1] DARPA Common Intrusion Detection Framework.
http://seclab.cs.ucdavis.edu/cidf/

[2] Feiertag, Rich, Cliff Kahn, Phil Porras, Dan Schnackenberg, Stuart Staniford-Chen, and Brian Tung, ed. "Common Intrusion Specification Language," DRAFT Specification, 8 June 1998. http://gost.isi.edu/projects/crisis/cisl_0.7.txt

[3] Concordia Mobile Agent System.
http://www.meitca.com/HSL/Projects/Concordia/Welcome.html

[4] Mixer, Robert. "A Network Security Agent Architecture." Texas A&M University, 1997.

[5] C Language Integrated Production System (CLIPS).
http://www.ghg.net/clips/CLIPS.html

[6] Computer Oracle Processing System (COPS).
ftp://coast.cs.purdue.edu/pub/tools/unix/cops/

[7] Java Remote Method Invocation API.
http://java.sun.com/products/jdk/1.2/docs/guide/rmi/spec/rmiTOC.doc.html

[8] Java Reflection API.
http://java.sun.com/products/jdk/1.2/docs/guide/reflection/spec1/java-reflection.doc.html

APPENDICES

Appendix A: Classes in this Distribution:

A-1. PACKAGE secure.parser

```
class secure.parser.Cache (implements java.io.Serializable)
interface secure.parser.IndexOwner
class secure.parser.DBInfoClass (implements secure.parser.IndexOwner,
    java.io.Serializable)
class secure.parser.DBInfoField (implements java.io.Serializable)
class secure.parser.SecurityFile (implements java.io.Serializable)
class secure.parser.IgnoreRule (implements java.io.Serializable)
class secure.parser.ContainsIgnoreRule
class secure.parser.StartsWithIgnoreRule
class secure.parser.EqualsIgnoreRule
class secure.parser.SubstringIgnoreRule
class secure.parser.ItemFilter (implements java.io.Serializable)
class secure.parser.DateItemFilter
class secure.parser.DirnameItemFilter
class secure.parser.FilenameItemFilter
class secure.parser.HostnameItemFilter
class secure.parser.SubstringItemFilter
class secure.parser.ItemRule (implements java.io.Serializable)
class secure.parser.ExternalItemRule
class secure.parser.GlobalItemRule (implements java.io.Serializable)
class secure.parser.InternalItemRule
class secure.parser.Keyword (implements java.io.Serializable)
class secure.parser.KeywordList (implements secure.parser.IndexOwner,
    java.io.Serializable)
class secure.parser.Line (implements java.io.Serializable)
class secure.parser.LineRule (implements java.io.Serializable)
class secure.parser.Parser (implements java.io.Serializable)
class secure.parser.ParserRules (implements java.io.Serializable)
class secure.parser.Queue (implements java.io.Serializable)
interface secure.parser.SingleStep
```

A-2. PACKAGE secure.parser.gui

```
class secure.parser.gui.WizardPanel
  class secure.parser.gui.DelimiterWizardPanel
  class secure.parser.gui.FileWizardPanel
  class secure.parser.gui.FinishWizardPanel
  class secure.parser.gui.IgnoreRuleWizardPanel
  class secure.parser.gui.ItemRuleWizardPanel
  class secure.parser.gui.LineRuleWizardPanel
  class secure.parser.gui.NameWizardPanel
  class secure.parser.gui.UpdateWizardPanel
  class secure.parser.gui.WelcomeWizardPanel
  class secure.parser.gui.HelpDialog
class secure.parser.gui.MainFrame
class secure.parser.gui.WizardPanelInfo
class secure.parser.gui.KeywordListNode
class secure.parser.gui.ParserWizard
class secure.parser.gui.TextAreaPipedInputStream
class secure.parser.gui.DBInfoClassNode
```

Classes in this Distribution (continued)

A-3. PACKAGE secure.service

```
interface secure.service.SecureDirectory
interface secure.service.SecurityDB
interface secure.service.SecurityToolInterface
  interface secure.service.CopsInterface (extends secure.service.SecurityToolInterface)
  interface secure.service.TigerInterface (extends
    secure.service.SecurityToolInterface)
class secure.service.CopsInterfaceService
class secure.service.TigerInterfaceService
class secure.service.DirectoryService
class secure.service.SecurityDBService
```

A-4. PACKAGE secure.db

```
class secure.db.DBClass (implements java.io.Serializable)
  class secure.db.Area
  class secure.db.Directory
  class secure.db.File
  class secure.db.Groups
  class secure.db.Host
  class secure.db.Other
  class secure.db.Problem
  class secure.db.ProblemDirectory
  class secure.db.ProblemFile
  class secure.db.ProblemGroup
  class secure.db.ProblemOther
  class secure.db.ProblemTime
  class secure.db.ProblemUser
  class secure.db.User
  class secure.db.UserGroup
```

A-5. PACKAGE secure.info

```
class secure.info.DBEntry (implements java.io.Serializable)
class secure.info.DBInfo (implements java.io.Serializable, java.lang.Cloneable)
  class secure.info.DirectoryInfo
  class secure.info.FileInfo
  class secure.info.OtherInfo
  class secure.info.ProblemInfo
  class secure.info.UserGroupInfo
```

A-6. PACKAGE secure.sql

```
class secure.sql.SQL
class secure.sql.SQLInsert
class secure.sql.SQLQuery
class secure.sql.SQLUpdate
```

A-7. PACKAGE secure.exception

```
class secure.exception.SecureException (implements java.io.Serializable)
class secure.exception.DBException
class secure.exception.DuplicateException
class secure.exception.FormatException
class secure.exception.NotFoundException
```

Appendix B: Software in this distribution

B-1. Tools

(1) Parser Specification Wizard: administrator develops the parsing rules that the parsers in the system will use

(2) CronAgent: an agent that functions similarly to a cron job by waking up at the specified interval and inquiring of the SecurityInterface services whether new data exists or not.

B-2. Accompanying Documentation

(1) API documentation: documentation all classes, methods, and field variables generated by JavaSoft's `javadoc` utility. This is very useful for future enhancements and maintenance.

(2) Help files: help files for the Parser Specification Wizard to give instructions for use. These are available from within the Wizard and by accessing the files in the distribution.

(3) Installation instructions: located in the root directory of the distribution.

(4) SQL data definition commands: a text file with the SQL data definition commands to establish the relational database on the database of your choice.